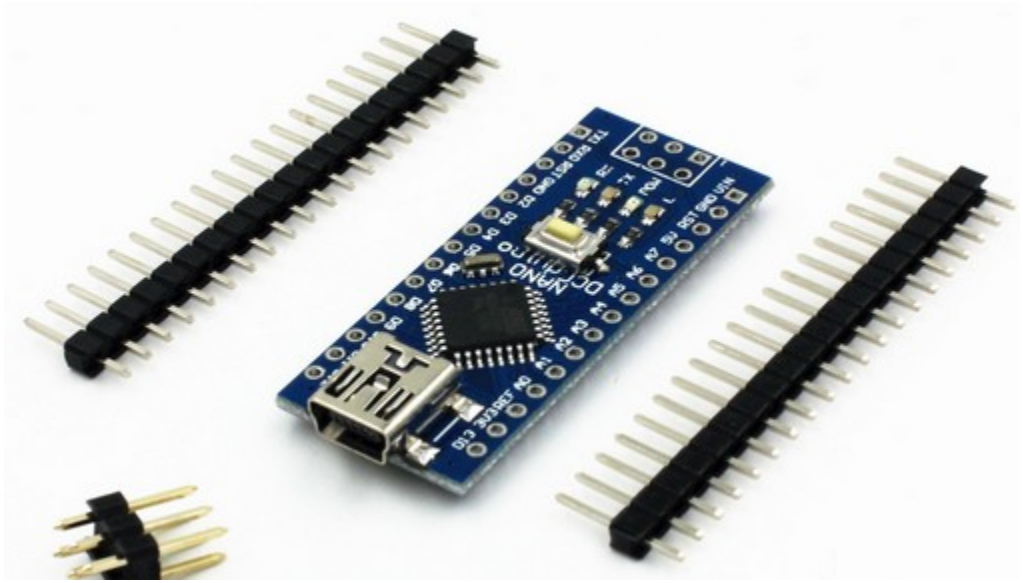
Rrr-duino for Model Railroading
a Hands-on approach!

Part 2



by Speed

(Gert 'Jim' Muller)

Introduction:

- For someone who has not seen Part 1 of this Clinic, we decided to do a **hands-on** or **make-and-take clinic** where everyone bought some parts: an Arduino Nano, USB cable, 9g Servo motor, breadboard, 3mm LEDs of many colors, some FETs a push button and a light sensor. (Oh and wires to connect things together)

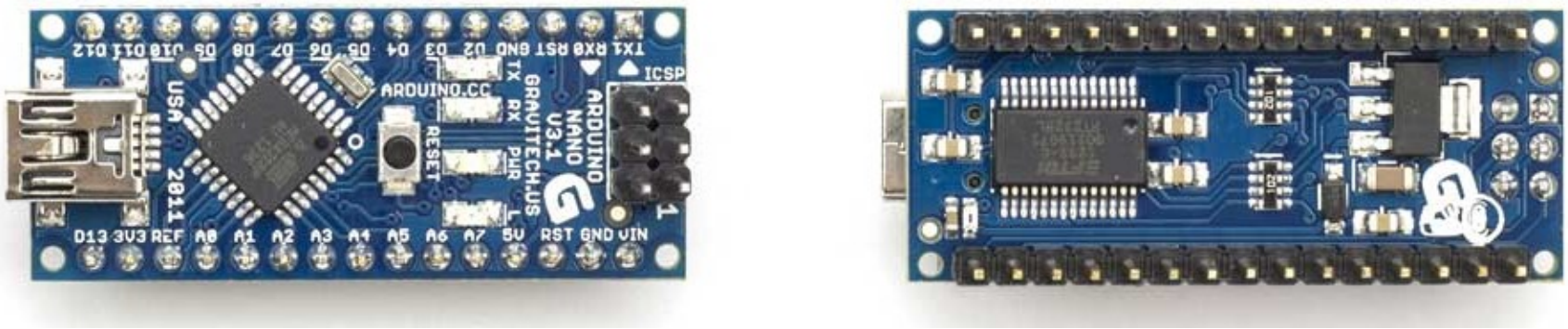
- What was never shown in the previous slides: Everyone installed the **CH340 device driver**, so the Nano shows up as **COMx** in **Device Manager** as well as the **Arduino Softw**are from arduino.cc

- What was also hinted in the previous slides: **Everything** we have done or emailed in the past, is shown on the **TxNamib** website: http://www.txnamib.com/electronics-and-software/rrrduino-for-model-railroads

Setup

Get your laptops up and running and plug the Nano in!

[PC → USB Cable → Nano]



*See a **solid** red light and a **blinking** red* light?*

Not? Then open File->Examples->01.Basics->Blink in the Software and upload it!

How 'bout now?

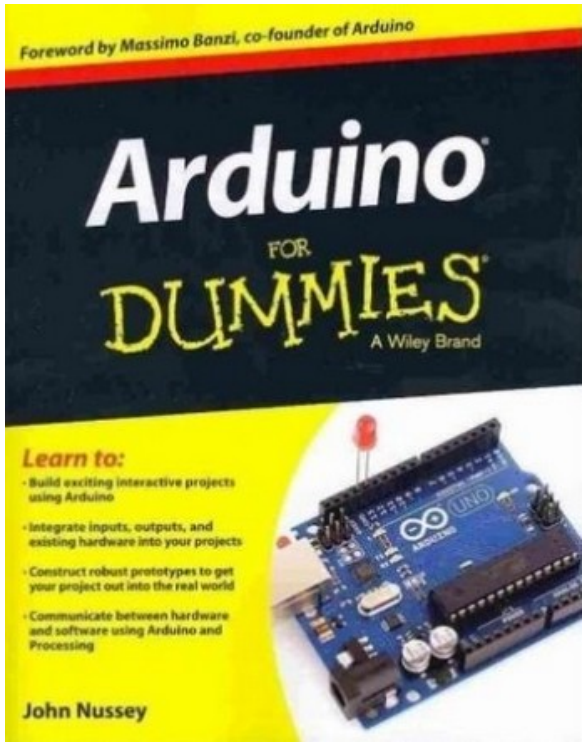
What is an Arduino?

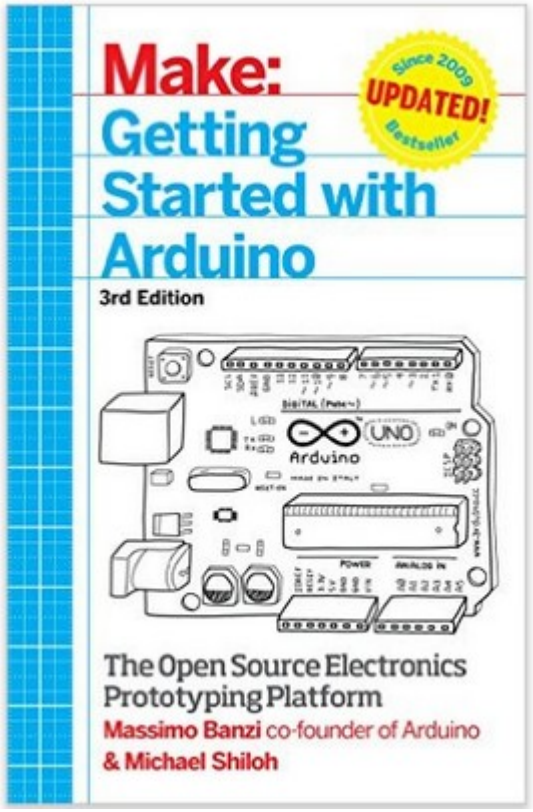
Quoted: (http://www.circuitstoday.com/story-and-history-of-development-of-arduino)

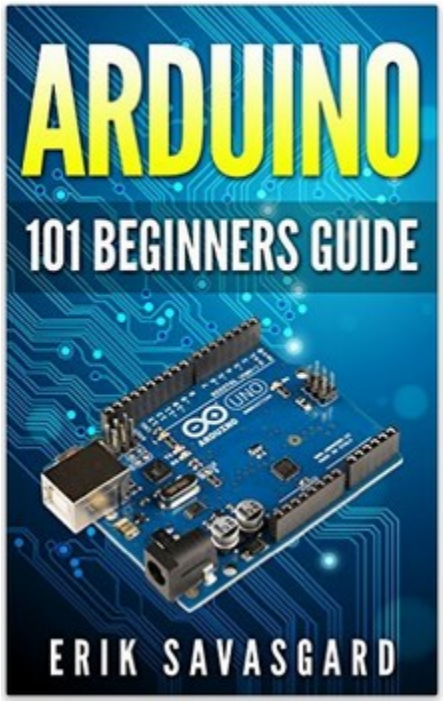
The new prototype board, the Arduino, created by Massimo Banzi and other founders, is a **low cost microcontroller board** that allows even a **novice** to do great things in electronics. An Arduino can be connected to all kind of **lights**, **motors**, **sensors** and **other devices**; easy-to-learn programming language can be used to program how the new creation behaves. Using the Arduino, you can build an **interactive display** or a **mobile robot** or anything that you can imagine.
You can purchase an Arduino board for **just about US $30** or **build your own** board from scratch. Consequently, Arduino has become the most powerful open source hardware movement of its time.

... but this was all said in Clinic Part 1, so lets move on!!!

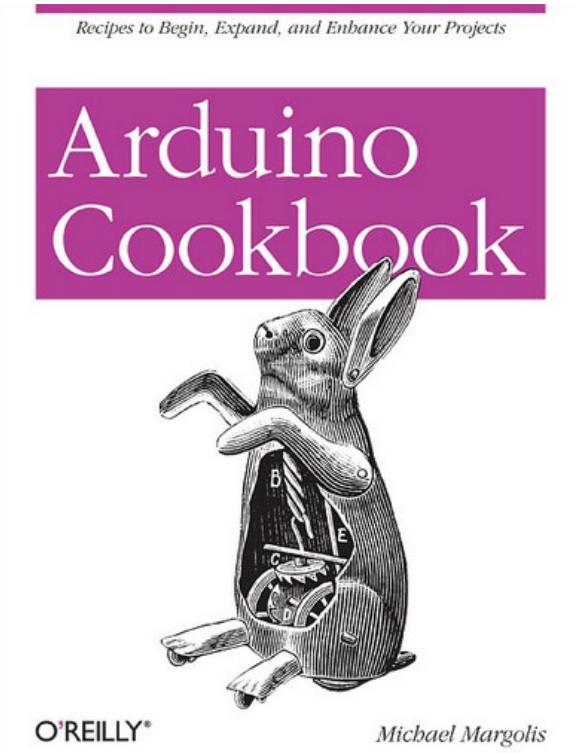
Things to Read (Easier):

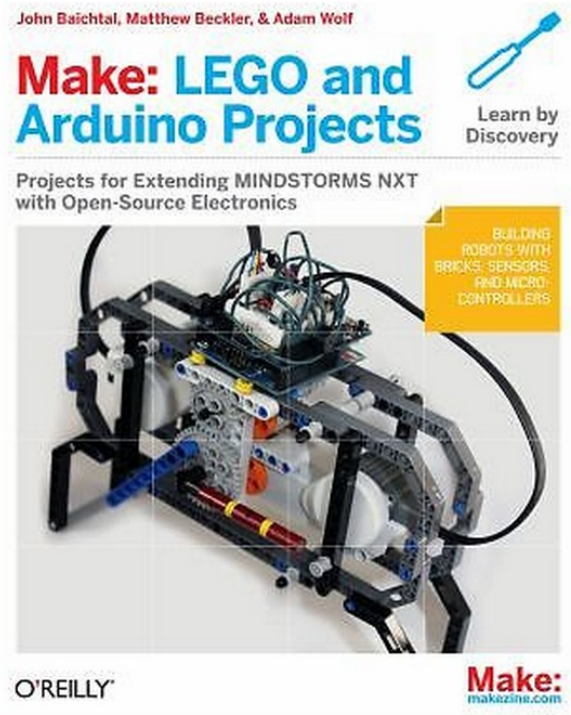


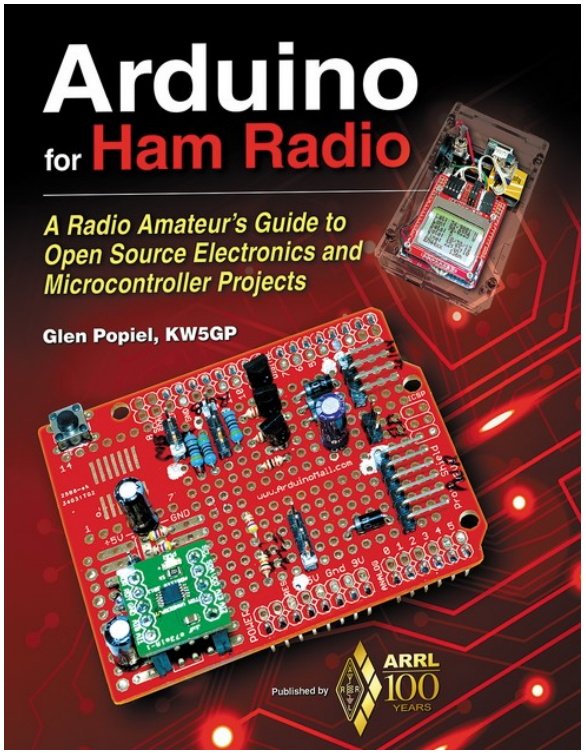




Things to Read (Harder):



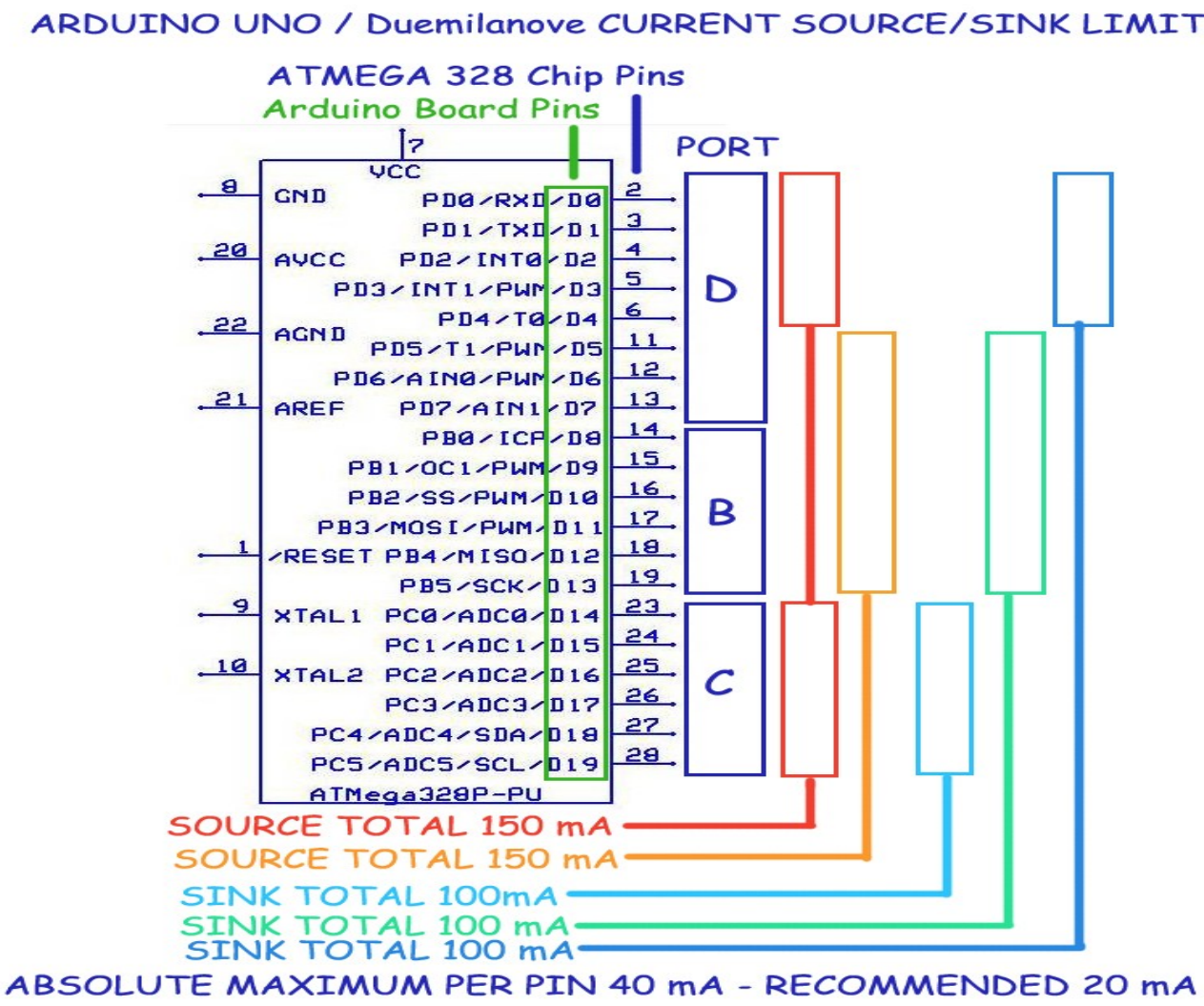




Arduino Nano

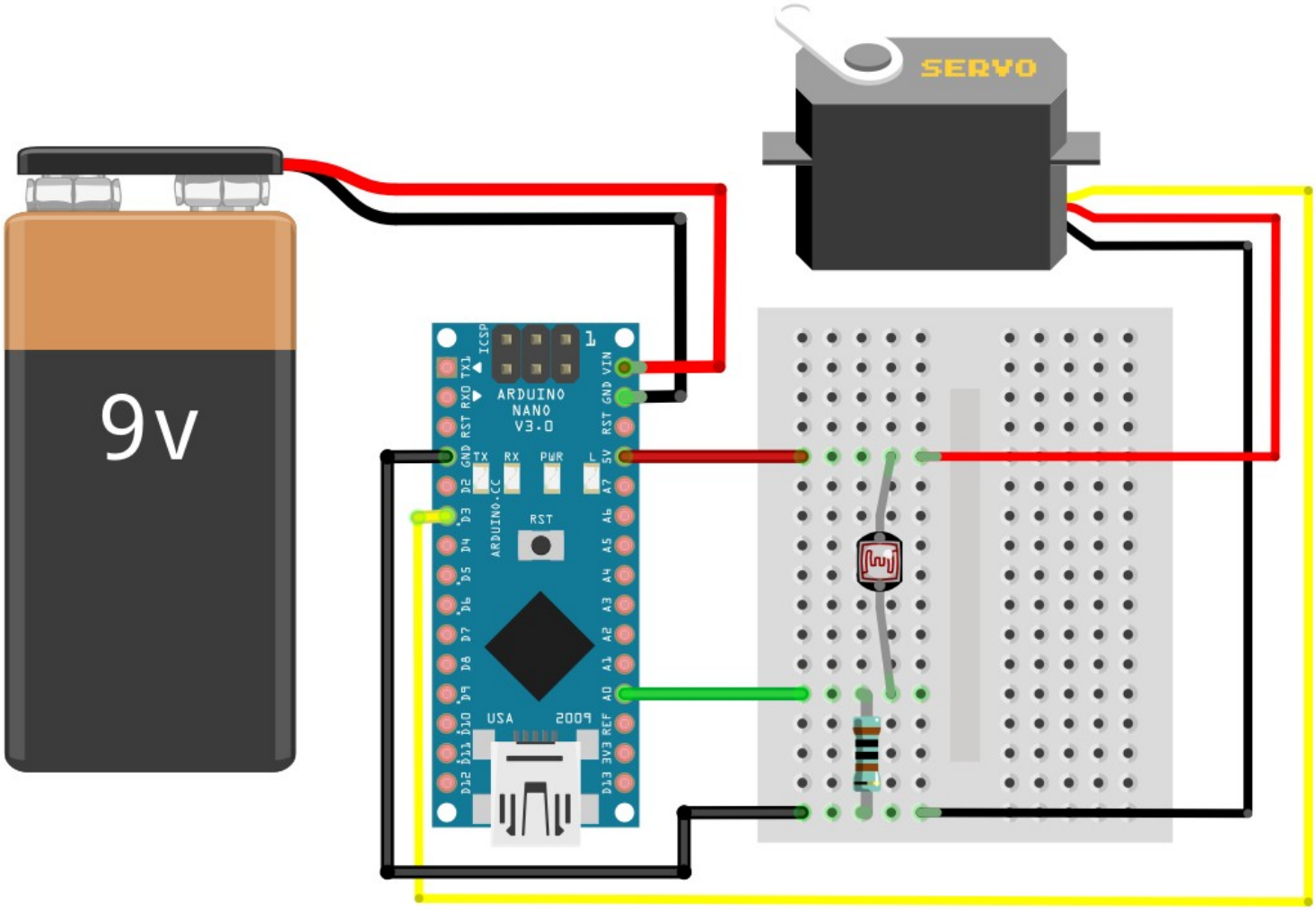
Microcontroller	**Atmel ATmega328**
Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	**14** (of which **6 provide PWM output**)
Analog Input Pins	8
DC Current per I/O Pin	40 mA ()
Flash Memory	**32 KB** of which 2 KB used by bootloader
SRAM	**2 KB**
EEPROM	**1 KB**
Clock Speed	16 MHz
Dimensions	0.73" x 1.70"
Length	45 mm
Width	18 mm
Weigth	5 g

A little more about Atmega328 “current” or mA:



http://playground.arduino.cc/Main/ArduinoPinCurrentLimitations

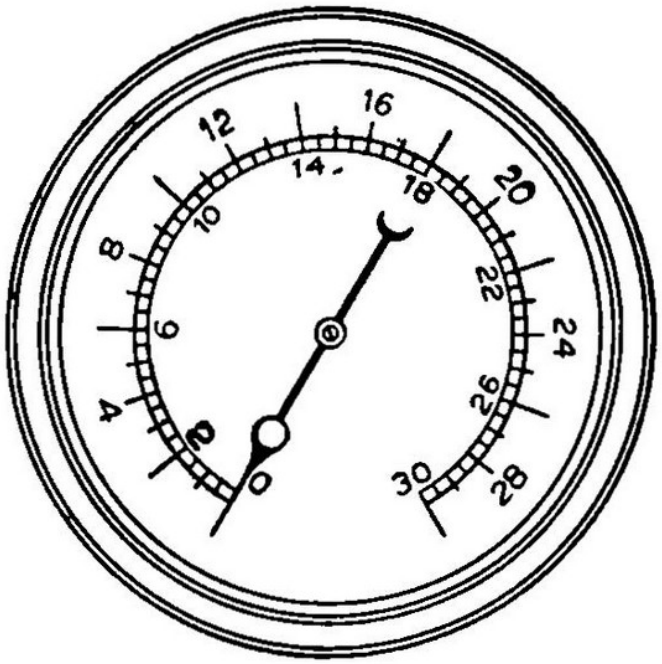
A Servo Following the Light!

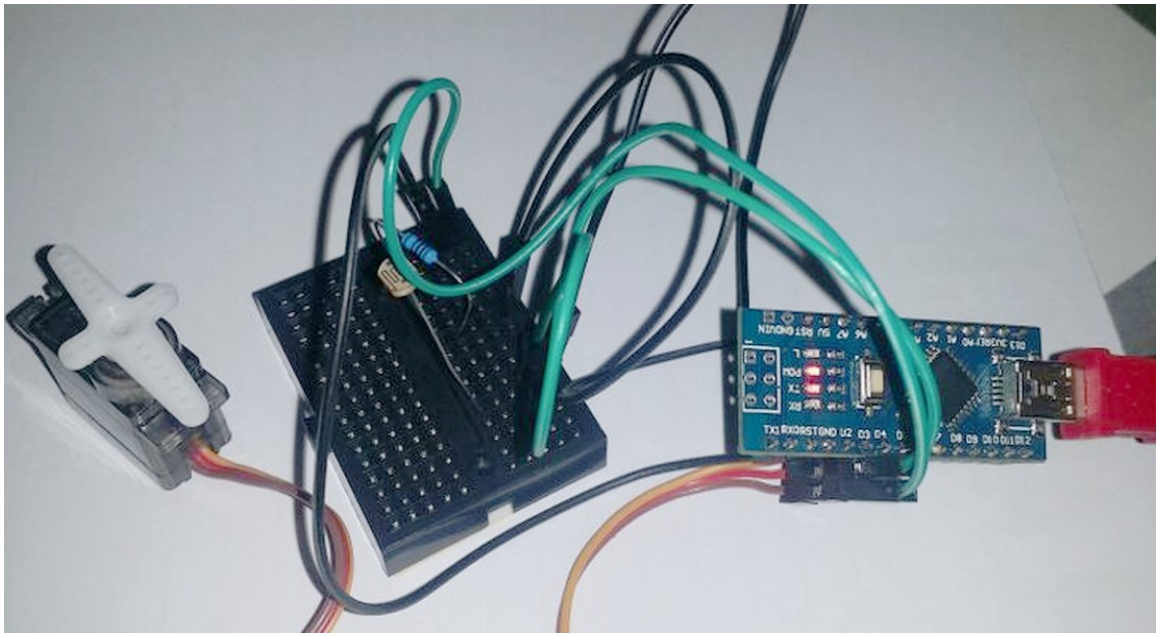


Servo Control to D3 (yellow) **Light Sensor to A0 (green)**

- Why did we talk about **mA in previous slide**? So you know how **many** 4mA **LEDs** you can drive (or sink) and that the **USB cable** might not bring enough **mA** to keep your **servo** happy.

Maybe you need to build a Dial of some sort?



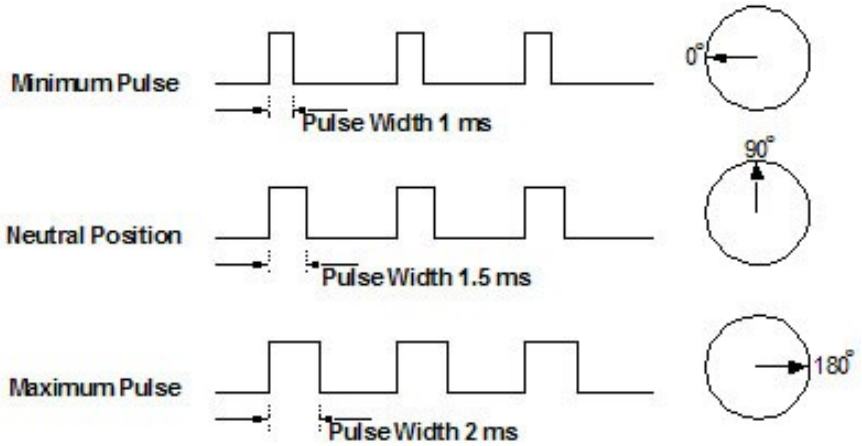


Servos?

- A servo motor has a built in control system to ensure that the position is maintained as commanded by sensing where it is and correcting any noticeable error.

- All the servos used in the model aircraft world, like our Micro 9g servo, is controlled by sending a pulse at 50 Hz (that is one pulse every 20 ms)

- And the width of the pulse tells the servo where to move to. (Also know that you get servos that can only 90, 180 or 360 degrees, and then continuous running ones where the speed of the servo is set buy the pulse width)



- And then there are some servos that can also move a little further when you give them a 0.5 ms pulse and/or further when they get a 2.5 ms pulse.

- The servo is also powered itself (5V in our case), and it must be able to provide enough current to maintain the servo's position and movement. Else it might jitter, or reset when told to move. Very common using a Nano on a 400mA USB

Code:

```
/*  lightServo.ino  */

#include <Servo.h>                    // include the servo library

Servo myServo;                        // create a servo object

int const anaPin = A0;                // analog pin used to connect the potentiometer
int const servoPin = 3;               // pin used to connect the servo

int lightVal;                         // variable to read the value from the analog pin
int angle;                            // variable to hold the angle for the servo motor

void setup() {
  pinMode( anaPin, INPUT );
  myServo.attach( servoPin );         // attaches the pin to the servo object
  Serial.attach( 9600 );              // serial port, baud rate;
                                      //   press Ctrl+Shift+M after upload
} // setup

void loop() {
  lightVal = analogRead( anaPin );    // read the value of the potentiometer

  angle = map( lightVal, 0, 1023, 0, 179 );   // scale the numbers from the sensor

  Serial.print( lightVal );           // show the value to us on the Serial port
  Serial.print( " -> " );
  Serial.println( angle );            // show the angle to us on the Serial port
  myServo.write( angle );             // set the servo position

  delay( 50 );                        // give the servo some time

} // loop
```

Coding: Formatting

- **Space** and **Tabs** do not matter (except when splitting words apart!)
- Making it **look nice**, **helps** reading and debugging it
- **Comments** gets removed, type more comments, even if you think you don't need it! /* for blocks */
- **#define** and **#include**, preprocessor directives
  - #include puts the text from the file specified right there, imagine a copy-paste

    ```
    #include <Servo.h>
    ```

  - #define will swop the next word with the text behind it

    ```
    #define myPIN 4 + 4
    ```

- Every (must have a), same with “”, '', {} and []
- { and } creates code blocks

```
if ( x == 1 )     doThis();
```

vs

```
if ( x == 1 )     {
     doThis();
     andAlsoDoThis();
}
```

- Every statement ends with '**;**'
- C and C++ are **CASE sensitive**, `printLn();` is **not** the same as `pRintln();`

Coding: Variables

- A variable holds a value (or a bunch of them), we also have different types, or kinds

```
int piggyBankCoins = 5;
float piggyBankDollars = 0.25;

char letter = 'a';
String txtMsg = "Hi there";
```

- Just like your piggy bank, storing coins, how much money is in it?
- They can usually change, like adding a quarter to the piggy bank.
- Unless

```
const int a = 5;  // can never change, unless text changed here and recompiled
```

- Scope? A variable declared **in a function or method** can only be used **IN** that function
- Since you are the developer, you get to choose what name to use for your variable!

```
a = b * c;  // is syntactically correct, but, its purpose is not evident.
```

Contrast this with:

```
weekly_pay = hours_worked * pay_rate;
```

- Naming conventions: CamelCase, Delimiter_in_use, etc..., pick one and stick to it!

```
int weekPay; int WeekPay; int decimalLocalWeekPay; int dlWeekPay;
```

- How to decide, simple: if you were to read this code again in 2 years, what would make it the easiest to follow?

Coding: Operators

- `int a = 5 * b;` `// there are also +, -, /, %, ++, –, &, |, ^, &&, || and !`
- `if ( a == b ){}` `// there are also !=, >, <, <=, >=`

Keywords

- `if, else, do, while, for, switch, case, default, break, goto, void, return`
- `types: void, int, float, double, char, register, enum, struct`
- `specifying: const, extern, volatile, long, short, signed, static, auto`

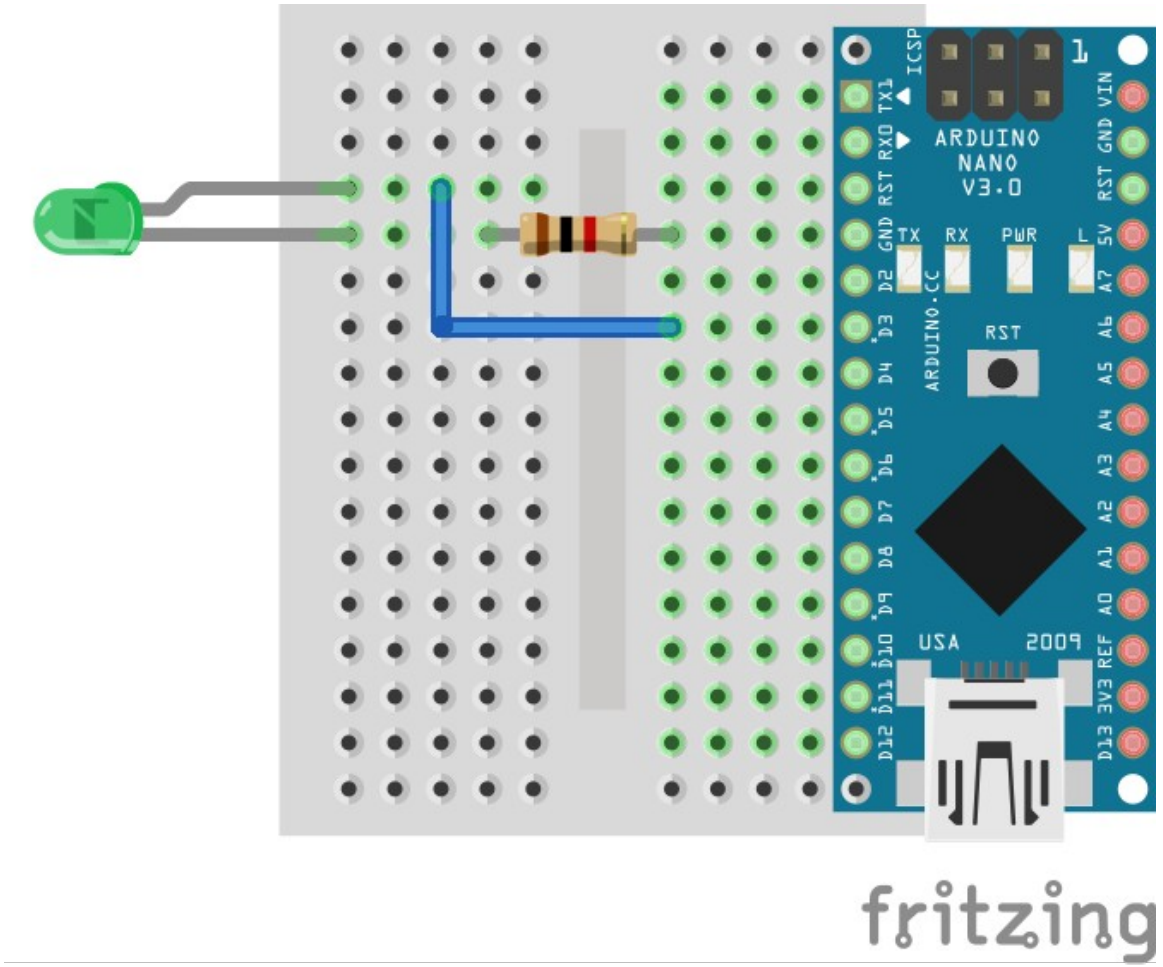
Short story, you can't use these as variable or function names!

Functions or Methods

- You have **seen**: `delay(x), pinMode(x,y), digitalWrite(x,y), map(a,b,c,d), random(x)`
- A **function** performs some **function**, right?
- It might take **zero or more** values to do so: `time = millis();` or `delay( 1000 );`
- It might give **a value back** `time = millis();` **or not** `delay( 1000 );`

```
int sumTheseUp( int x, int y ) {
  return x + y;
} // sumTheseUp

void setup() {
  pinMode( sumTheseUp( 1, 12 ), OUTPUT );
} // setup
```

A Serial Port and EEPROM using a Function!



- The EEPROM provides a way to store values between power off and power on.
- 1024 (8-bit) bytes on ATmega328
- Just like the SD Card in your camera, except this storage is already inside the Atmel chip! No card needed.

A Serial Port and EEPROM using a Function (Code)!

```
/*  serialEEPROMLed.ino  */
#include <EEPROM.h>                     // include the EEPROM library
int ledPin1      =  3;
int incomingByte =  0;                  // for incoming serial data

void setup( ) {
  pinMode( ledPin1, OUTPUT );           // set as an output
  Serial.begin( 9600 );                 // opens serial port, sets data rate to 9600 bps
  analogWrite( ledPin1, EEPROM[ 0 ] ); // output EEPROM value to ledPin1, using PWM
} // setup

void loop( ) {
    checkIncoming( );                   // check incoming serial data
} // loop

void checkIncoming( ) {
  unsigned int answer = 0;
  if ( Serial.available( ) > 0  ) {     // check serial data
    incomingByte = Serial.read( );      // read the incoming byte

    if ( incomingByte == 'v' ) {
      Serial.print( "You pressed 'v'!" );
    } else if ( ( incomingByte >= '0' ) && ( incomingByte <= '9' ) ) {
          answer = ( incomingByte - 48 );  // zero based
          EEPROM[ 0 ] = 10 * answer;
          analogWrite( ledPin1, EEPROM[ 0 ] );
      } else {
        Serial.print( "I received: " );     // say what you got:
        Serial.println( incomingByte, DEC );
    } // else
  } // if serial available
} // checkIncoming
```

Multitasking!

- We would like to use more than one pin on the Arduino to do something. Yes there are 20 pins to use.
- If we use `delay( 1000 );` to wait for the next Yellow traffic light to change to Red, then the Campfire, the Welder, the Grade Crossing, and the Servo moving based on the light value, ALL have to wait 1 second too! And that is going to make welding and campfire look pretty. Insert a `delay( 1000 );` at the end of the Welder code and see if you like it?
- What if we could ask what the current time is and decide if we need to do something or not? So, we calculate how much time has expired since the last change, and if we exceed the threshold, we change!
- Please meet millis();

Description

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

```
#define CAMPFIREUPDATE          100
#define TRAFFICUPDATE          1000
Long int lastCampfireChange     = 0;
long int lastTrafficLightChange = 0;

void loop() {
  long int now = millis();   // get milliseconds since Arduino started
  if ( ( now - lastTrafficLightChange ) > TRAFFICUPDATE ) {
    if ( tLight == HIGH ) tLight = LOW; else tLight = HIGH;
    digitalWrite( TPIN, tLight );
    lastTrafficLightChange = now;
  } // if time for traffic to update
  if ( ( now - lastCampfireChange ) > CAMPFIREUPDATE ) {
    … //  lastCampfireChange = now;
  } // if time for campfire to update
} // loop
```

OOP (Object Orientated Programming)?

- Code and data are usually separate, you write functions and they act on the data you pass along, but what if I could store a method with my data? Don't worry, you have already used this,

```
Servo myServo; myServo.attach( pin ); myServo.write( 50 );
```

- A very simple way to show the use of this, is to try to avoid using delay() and use millis() instead, but inside an object that contains its data.

- OOP is not needed to do this, but it makes things a lot cleaner and easier to reuse your code somewhere else. On the previous slide, you need to repeat the code for every Campfire LED attached to every pin.

- We could write code in the loop function to check millis() every time coming through, if a green light needs to be on, or if a yellow light needs to be one, or a red one, and, at the same time, if our campfire light needs to turn on or off.

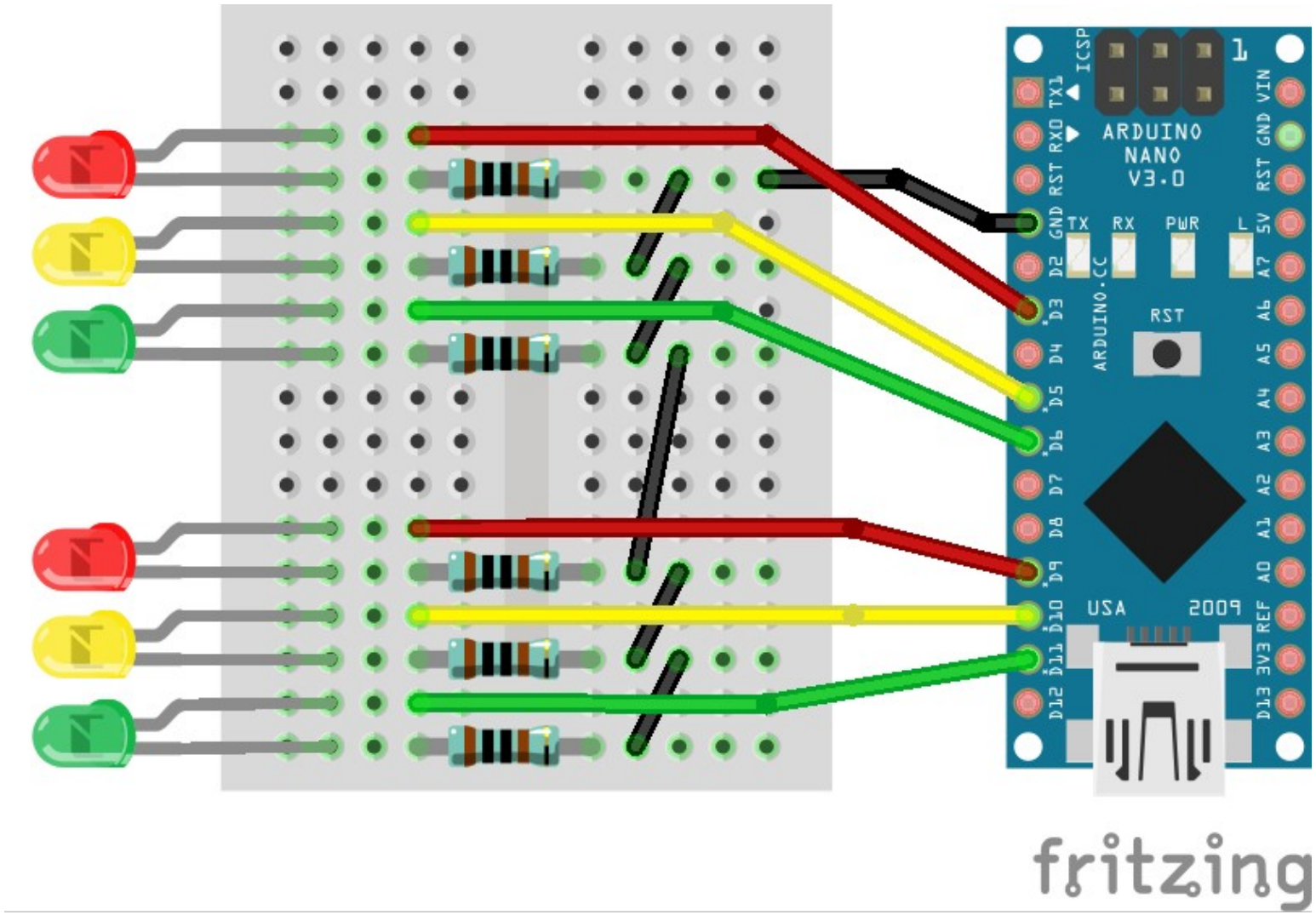
```
CampFire myFire( firePin );
TrafficLite myLight( redPin, greenPin, yellowPin, speed );

void setup() {
 // nothing need here, unless you need to want to do the pinModes here
 // but the constructors for the objects will take care of that this time
} // setup

void loop() {
  myFire.update();
  myLight.update();
} // loop
```

```
class TrafficLite {
  int redPin;      // the number of the red LED pin
  public:
  // Constructor - creates a TrafficLight
  // and initializes the member variables and state
  TrafficLite( int rPin, int yPin, int gPin ) {
    redPin = rPin;
    pinMode( redPin, OUTPUT );
    output();
    previousMillis = millis();
  } // TrafficLite constructor

  void update() {
    // check to see if it's time to change the state of the LED
    unsigned long now = millis();
    if ( now - previousMillis >= UPDATETIME ) {
      previousMillis = now;
      if ( countDown  > 0 ) {
        countDown--;
      } else {
        ledState--;           // 4 goes to 3 only once, so no need to switch case 4:
        switch ( ledState ) {
        case 0:
          ledState = 3;
        case 1:
          countDown = RED_TIME;
          break;
…
…
…
        }; // switch
        output();                    // Update the actual LEDs
      } // time for change yet?
    } // if
  } // Update
}; // TrafficLite
```

- Complete example in TrafficLight_OOP.ino, showing two traffic lights:

```
/* TrafficLight_OOP.ino */

//
// Class code here with update() and other methods
//

TrafficLite lite1( NS_REDPIN, NS_YELPIN, NS_GRNPIN );
TrafficLite lite2( EW_REDPIN, EW_YELPIN, EW_GRNPIN );

void setup() {
  lite1.go();
  firstTime = 1;
} // setup

void loop() {
  if ( firstTime == 1 ) {
    if ( lite1.isRed() ) {
      lite2.delayStart();
      firstTime = 0;
    } // if isRed
  } // if firstTime
  lite1.update();
  lite2.update();
} // loop
```

Traffic Light – OOP



Questions?

References:

- http://www.chemistrylearning.com/writing-c-program/
- http://www.kenleung.ca/_portfolioassets/PDF/HistoryOfArduino_KenLeung.pdf
- http://www.circuitstoday.com/story-and-history-of-development-of-arduino
- www.Arduino.cc
- http://sumidacrossing.org/LayoutElectricity/Arduinos/ArduinoShields/
- http://opendcc.sourceforge.net/
- http://arduino.cc/en/Guide/ArduinoGSMShield
- http://www.martyncurrey.com/arduino-nano-as-an-isp-programmer/
- https://www.servocity.com/html/how_do_servos_work_.html#.VhheI_lViko

Arduino Nano Pins, reference:

- Purple (and green) numbers used in software, as labeled on edge of board.
- Grey numbers are the Atmel chip's pin numbers, if you were to use the chip itself somewhere else!